Project Report

1. Abstract

Audiobook maker is a web application that converts text input from the user to a speech using Amazon Polly. Not all the physical books are converted to an audiobook for several reasons. This web application allows users to convert their favourite books into an audiobook.

The process of converting text to speech runs asynchronously, hence if a user wants to upload another file just after the first upload, they don't have to wait for the first process to complete. This way the website doesn't get stuck after an upload even if the uploaded file contains large number of characters. The user will receive the URL of speech file on their email once the processing is finished. It might take a while to receive the email if the input file is large.

The current state of the application should be able to serve at most 1000 concurrent users at once without any impact on performance.

2. AWS Services

- **Compute:** AWS Elastic Beanstalk & AWS Lambda

  The primary reason to choose AWS Elastic Beanstalk is to utilise the out of box auto-management of EC2 instances, scaling, load balancing and monitoring. Due its less operational overhead this seemed the best choice out of all the other options for compute. It also automatically handles OS updates and patches.

  AWS EC2 requires manual configuration (Scaling Policies, Auto scaling group, Launch Template etc.) to auto scale.

  AWS Lambda is not suitable for hosting the front-end, so it was used for performing task specific operations. This also helps in reducing costs associated with lambda functions as they are billed only for the number of invocations and the amount of time they run, which in this case is very small as the speech conversion process is asynchronous.

  AWS Batch is meant for background jobs especially large-scale batch processing and not for hosting. **AWS Fargate** charges per memory/CPU usage and it would increase the cost because the hosted application must stay up always and would cause CPU to be always busy. While the **AWS lightsail** would also have been a good choice but it lacks the flexibility of Beanstalk for automatic scaling.

- **Storage:** Amazon Simple Storage Service (S3)

  The primary reason for choosing S3 for storing the input text files and output speech files is that it automatically scales to handle large amount of data, and the user doesn't have to define any

limit or quota of storage. Moreover, it allows direct web access to the stored objects through URLs, and it integrates well with other AWS services.

Other services: **AWS Backup** is meant for backup of AWS resources and not for storing files. Because we would need constant access of the objects within S3, if the files were to be stored on AWS Backup the cost would rise substantially. **AWS Elastic block** store requires an EC2 instance to function and we don't want our storage to be dependent of any instance hence S3 is the better choice.

AWS Elastic File System costs higher per GB than S3, and **AWS Elastic Disaster Recovery** is meant for replicating workloads to recover from failures, which makes them unsuitable for our purpose. AWS File Cache, as the name suggest, is useful for temporary storage and not persistent storage.

- **Networking and Content Delivery:** Amazon API Gateway

  Amazon API gateway allows using REST endpoints to perform specific operations. In this application the API gateway's purpose is to fetch a presigned URL of the S3 bucket using a lambda function. We need a new presigned URL for each new upload, hence fetching it using the API gateway's REST endpoints makes more sense.

  AWS CloudFront is a service for caching content delivery, and it doesn't handle REST API requests. **AWS Route 53** manages domain names and routing but doesn't trigger a lambda or handle HTTP requests. AWS verifies access is for securely connecting private apps, not for public API exposure. **AWS VPC** is used by default when we create resources, so we still need API gateways to expose endpoints.

  The other services manage distribution of traffic and securely access other resources but API gateway is specifically built for exposing API endpoints to users.

- **Database:** Amazon DynamoDB

  DynamoDB is serverless, fully managed service, which is the best option for this application as there is no complex database setup required for complex relationship between data, and only the user logs would be stored in a single table. Not only that but automatically scales based on usage and as the operations to the table are constant time read/writes of user logs it is fast. Not only that but it only costs USD 1.25 per million writes and USD 0.25 per million read requests which would still cost a lot less even if there are billions of users of this application making multiple requests.

  AWS aurora is good for complex relational high-performance workloads but this app only requires a single table to store key-value pairs. **AWS ElastiCache**, as the name suggests, is good for storing temporary data but not when we want to keep user's progress persistent. **AWS Keyspace** would also be an overkill for a key-value type storage. Also, it requires extra setup to integrate with lambda and other AWS services. **AWS MemoryDB** stores data in RAM, hence it is volatile and costs more. Amazon Neptune is suitable for graph like data (such as social

connections etc.) and requires Gremlin or SPARQL which adds unnecessary complexity for our simple application.

Amazon RDS requires managing database instances while DynamoDB scales automatically. Moreover, RDS charges for instances, storage, backups and maintenance, whereas DynamoDB is pay-per-request. As we see, most of the database are overkill for our simple usage, hence DynamoDB is a better choice.

- **Application integration:** Amazon SES (Simple Email Service)

  Using Amazon SES, the application sends email to the user who requested an audiobook.

  AWS Step Functions, Amazon AppFlow, AWS B2B Data Interchange, Amazon EventBridge, Amazon Managed Workflows for Apache Airflow, Amazon MQ do not support sending emails by themselves. While Amazon Simple Notification Service (SNS) is made for sending emails, it is great for broadcasting messages to multiple subscribers at once. While one can achieve sending individual emails using SNS too but it requires creating new SNS topic any time a user requests for a book and it is not logical to use when we have SES.

- **Management and Governance:** Amazon CloudWatch

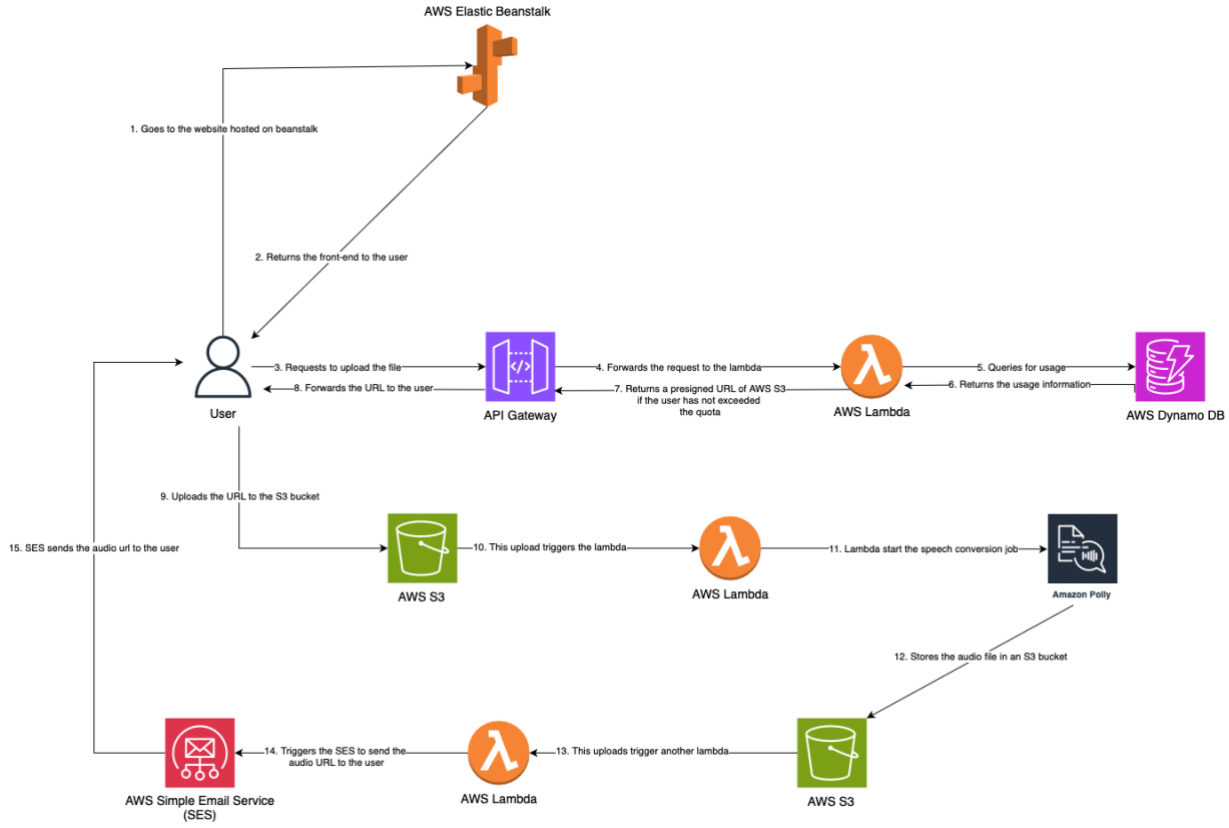
  CloudWatch enable lambda functions and other services to print logs for better debugging.

  AWS chatbot doesn't support logs. **AWS CloudTrail** records API activities such as who made changes but doesn't capture application logs. **AWS Compute Optimizer** supports cost and performance recommendations. Amazon Managed Grafana integrates with CloudWatch dashboard but does not store the log itself. Other services such as **AWS Trusted Advisor**, AWS Systems Manager, **AWS Service Catalog** etc. doesn't provide log monitoring as they are built for other functionalities.

3. Delivery Model

This project uses Software as a Service (SaaS) delivery model. If a user wants to create an audiobook out of a text file, they wouldn't want to build all or even part of the components that do this process by themselves. They would only want to upload a file and get the output. This saves users valuable time. Hence, Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) is not suitable for this task.

4. Architecture



4.1 Architecture

How do all of the application components fit together to deliver your application?

The user interacts with the front-end of the application, hosted on EC2 instance, using Elastic Beanstalk's public URL. When the user submits the input file and email as an input to the API gateway, a lambda function is triggered which sends a pre-signed URL of an S3 bucket if the user has not exceeded their quota. Quota checking is done by querying the DynamoDB.

Front-end then stores the text file in the input S3 bucket using the pre-signed URL. This triggers another lambda function that calls Amazon Polly to asynchronously convert the text into a speech. Once the conversion has been completed, Polly stores the output in the output bucket.

When the .mp3 file is stored in the output bucket, another lambda is triggered. This lambda sends the publicly accessible URL of the object to the email provided by user at the time of the input with the help of AWS Simple Email Service (SES).

Once the user receives the URL, they can use that URL to download the audiobook.

Where is data stored?

The data is stored in two places. The input and output text files are stored in their respective S3 buckets, while user's email ID and the number of requests they have made are stored in the DynamoDB to serve the purpose of quota check.

What programming languages did you use (and why) and what parts of your application required code?

I used JavaScript to implement Lambda functions and the front-end code. I used JavaScript because I am familiar with building apps using react.js. I could have used TypeScript as well because it provides type checking, but the app's scope is quite small, and I handled errors using various try-catch blocks. Moreover, all the AWS services are accessible using AWS libraries built for JavaScript.

How is your application deployed to the cloud?

AWS Elastic Beanstalk: The front-end code is deployed on the EC2 instance created by the Elastic Beanstalk. Elastic beanstalk automatically handles the deployment of the application which includes building an image from the DockerFile and starting a container using that image. Elastic beanstalk is configured with Docker runtime.

AWS Lambda: The back-end code that involves checking user's quota from DynamoDB, triggering text-to-speech conversion and sending emails to the users is deployed on various lambda functions. This ensures that the important back-end code is secured, and nobody can access the database or the lambda functions from any place other than the front-end.

Solution Architecture

The architecture of this application uses AWS services such as Elastic Beanstalk, Lambda Functions, DynamoDB, Polly, S3 and SES. All of these services are native to AWS and doesn't require any external services.

5. Security Measurements

Services provided by AWS such as Lambda, EC2, DynamoDB are inherently secure and provide protection to data at rest and in transit. DynamoDB is a fully managed AWS service that does not run inside the VPC. Only resources with required permissions can access a particular DynamoDB table.

Currently, only the services with certain roles can access other services. This provides a bit of security to the services.

Although the cloud is secure, the project can make use of private subnets to make the architecture robust. We can configure the API gateway and the lambda functions to be inside of the private subnet so that only the EC2 instance running in a public subnet can access those services. This way

the back-end would be only accessible by the front-end, and nothing else. The current implementation would be accessible by anyone if they had the link to the API gateway. This is a vulnerability in the current architecture.

6. Cost to build the system from scratch

Firstly, to implement the current architecture on-premise, the organization would have to purchase many hardware equipment.

- **Servers:** Servers to host the web application.

  Estimated cost: A server with 32 GB RAM, 500 GB SSD and a 3.5 GHZ processor would cost around $1718 CAD. Moreover, monthly cost of power, cooling, and other expenses would be $30.24 CAD. [1]

- **Storage:** Storage device such as SSD or HDD will need to be purchased for input and output file storage, as well as the logs data. The demand for this will keep increasing as the usage increase. Hence, more storage devices will need to be bought as data increases.

  Estimated cost: Anywhere between $16 - $100 CAD per unit (128GB SSD)

- **Networking devices:** Routers, switches and firewalls will be required to provide connectivity between different components.

  Estimated cost: $1000 - $5000 CAD (depends on the business requirements). [2]

- **Operating system:** Each server will require an Operating System to run services.

  Estimated cost: No charge (Linux is open source)

- **Manpower:** A team of skilled engineers will be required to configure and ensure the smooth operation of the infrastructure.

  Estimated cost: $64000 CAD. [3]

Total estimated cost:

The **first month** cost would be: $2765 + the cost of equipment transportation + maintenance costs + employees salary.

After than the cost would increase if new equipment were purchased. If the workload is steady, then the cost would reduce to around, $200 (internet, other service charges) + maintenance cost + employees salary per month.

In the case of a disaster, disaster recovery procedure would cost the same as the cost of additional infrastructure.

7. Future development

I would improve the system to process the input in batch so that more than 1,00,000 characters can be converted to speech to accommodate large books. This would require text to be split into different chunks of size less than or equal to 1,00,000 characters, saving the total chunks and unique id to DynamoDB and then calling AWS Polly for each chunk. A lambda function will increment the count associated with the unique id in dynamo DB as each chunk is received in the form of speech and once that count equals to the total chunks count, it would notify the app running on AWS Fargate through SNS. AWS Fargate would fetch all the audio files and merge them to produce a final output file. Once that file is stored in the output bucket, a lambda function would get triggered and send the link of that file to the user's email address. Also, I would allow users to upload files in any language and then use AWS translate to convert it into the language they desire so that they can listen to the audiobook in the language of their choice.

8. References

[1] "How Much Does a Server Cost For a Small Business in 2025?", *Server Mania*. [Online] Available: How Much Does a Server Cost For a Small Business in 2025? [Accessed: March 01, 2025]

[2] "Small Business Network Setup Cost: Pricing Guide (2024)", *The Network Installers*. [Online] Available: Small Business Network Setup Cost: Pricing Guide (2024) [Accessed: March 06, 2025]

[3] "Software Developer Salaries", *'GLASSDOOR'*. [Online] Available: Software Developer Salaries [Accessed: March 08, 2025]